3D Electromagnetic Monte Carlo Particle- in-Cell Simulations on MIMD Parallel Computers

J. Wang[*], P. C. Liewer[†] and E. Huang[‡]

Jet Propulsion Laboratory

California. institute of Technology

Abstract

A three-dimensional electromagnetic particle-in-cell code with Monte Carlo collision (PIC-MCC) has been developed for MIMD parallel supercomputers. This code uses a standard relativistic leapfrog scheme incorporated with Monte Carlo calculations to push plasma particles and to calculate collisions.t effects on particle orbits. A local finite-difference time-domain method is used to update the self-consistent electromagnetic fields. The code is implemented using the Generat Concurrent PIC (GCPIC) algorithm which uses a domain decomposition to divide the computation among the processors. Particles must be exchanged between processors as they move among subdomains. We evaluate the performance of this code using a 512-processor Intel Touchstone Delta, a 512-processor Intel Paragon, and a 256-processor Cray T3D. It is shown that our code runs with a high parallel efficiency of $>95\%$ for large size problems. We have run PIC-MCC simulations using 162 million to 361 million particles with several million collisions per time step. For these large scale simulations, the particle push time achieved is in the range of 100-115 nsecs/particle/time step, and the collision calculation time in the range of a few hundreds nsecs per collision. Compared with the performance on a single CPU Cray C90, this represents a factor of 70-80 speedup.

1. Introduction

Computer particle simulation has become a standard research tool for the study of non-linear kinetic problems in space and laboratory plasma physics research. Collisionless plasma phenomena may be studied using a particle-in-cell (PIC) code. A PIC code models a plasma as hundreds of thousands of test particles and follows the evolution of the orbits of individual test particles in the self-consistent electromagnetic fields[1, 2]. Each time step in a PIC code consists of two major stages: the *particle push* to update the particle orbits and calculate the new charge and/or current density, and the *field solve* to update the electromagnetic fields. Since the particles can be located anywhere within the simulation domain but the macroscopic field quantities are defined only on discrete grid points, the *particle push* uses a "gather" step to interpolate fields from grid points to particle positions and a "scatter" step to deposit the charge/current of each particle to grid points. To study those problems that involve collisions between plasmas and neutral atoms, a particle-in-cell with Monte Carlo collision (PIC-MCC) can be used. In a PIC-MCC code, a Monte Carlo scheme is incorporated into the particle *push* stage of a PIC code to calculate the collisional effect on plasma particle orbits[3].

While the particle simulation method allows one to study the plasma phenomena from the very fundamentat level, the scope of the physics that can be resolved in a simulation study critically depends on the computational power. The coroputationat time/cost and computer memory size restricts the time state, spatial scale, and number of particles that can be used in a simulation. The cost of running three dimensional particle simulations on existing sequential supercomputers limits the problems which can be addressed.

Recent advances in massively parallel supercomputers have provided computational possibilities that were previously not conceivable. To illustrate the state-of-the-art of parallel supercomputers, in Table 1 we summarize the hardwares of the three MIMD (Multiple Instruction Multiple Data) parallel supercomputers op-

[*] Member Technical Staff, Advanced Propulsion, Member AIAA

[†] Member Technical Staff, Space Physics

[‡] Member Technical Staff, Supercomputing

crating at Caltech and the Jet l'repulsion Laboratory. The Intel Touchstone Delta has 512 numerical nodes each with a memory of 4 Mwords and a peak speed of 60 double-precision (75 single-precision) Mflops. The Intel Paragon has 512 numerical nodes each with a memory of 8 Mwords and a peak speed of 75 double-precision (100 single-precision) Mflops. The Cray T3D at JPL has 256 numerical nodes each with a memory of 8 Mwords and a peak speed of 150 Mflops. hence, the total memory size for Delta, Paragon, and Cray T3D are 2.048 Gwords, 4.596 Gwords, and 2.048 Gwords respectively and the total peak speed 30.7 Gflops, 38.4 Gflops, and 38.4 Gflops respectively.

In a previous study, we have developed a three-dimensional electromagnetic PIC code for MIMD parallel supercomputers[4]. That code uses a relativistic particle push, a local finite-difference time-domain solution to the full Maxwell's equations, and the General Concurrent PIC algorithm (GCPIC) which uses a domain decomposition to divide the computation among the processors[5]. The 3D EM PIC code has been implemented on the 512-processor Intel Delta, and has proven to be very efficient. (The efficiency is $\geq 95\ \%$ for problems with $\geq 2\times10^5$ particles/processor[4].)

In this paper, we extend the previous work to a 31) EM PIC-MCC code and implement this code on a 512-processor Delta, a 51 2-processor Paragon, and a 256-processor Cray T3D. In a simulation study, it is necessary to obtain benchmark information such a.. the computing speed and the simulation scale so one can decide the feasibility and effectiveness of applying PIC-MCC in a particular application. Hence, another objective of this study is to use the resulting 3D EM PIC-MCC code as a tool to benchmark the performances of the Delta, Paragon, and T3D for very-large-scale PIC-MCC simulations.

This paper is organized as follows: Section 2 discusses the algorithm used in our 31) EM PIC-MCC code and the parallel implementation; Section 3 analyzes the code performances on the Delta, Paragon, and Cray T3D; and Section 4 contains a summary and conclusions.

2. A Parallel 31) Electromagnetic PIC-MCC Code

The Algorithm

An electromagnetic PIC code solves Maxwell's equations for the macroscopic field and Newton's second law for individual particle trajectories:

$$\nabla\cdot\vec{E}=\rho \qquad (1)$$

$$\nabla\cdot\vec{B}=0 \qquad (2)$$

$$\frac{\partial\vec{E}}{\partial t}=c\nabla\times\vec{B}-\vec{J} \qquad (3)$$

$$\frac{\partial\vec{B}}{\partial t}=-c\nabla\times\vec{E} \qquad (4)$$

$$\frac{d\gamma m\vec{V}}{dt}=\vec{F}=q(\vec{E}+\vec{V}\times\frac{B}{c}),\quad \frac{d\vec{x}}{dt}=\vec{V} \qquad (5)$$

where relativistic effects are included in eq(5) ($\gamma=1/\sqrt{1-V^2/c^2}$). To include collisions between plasma particles and neutral atoms, one may include Monte Carlo collsion calculation into a PIC code, i.e. a PIC-MCC code (See Ref[3] for a review on PIC-MCC code).

In our code, the electromagnetic field equations are solved using a charge- conserving finite-difference leapfrogging scheme, which was used by Sandia National Laboratories in the Quicksilver code [7] and by Buneman et al in the Tristan code [8]. We choose this method (which is a "local" method) over transform methods (which are "global" methods) because a method that updates the field purely from the local data runs much more efficiently in parallel [4]. In this scheme, one obtains the current flux through every cell surface within a time step dt, $dt\vec{J}^{n+1/2}$ using a rigorous charge conservation method for current deposit, which is described in detail in [8]. Next, the electromagnetic field is updated *locally* by finite-difference leapfrogging in time:

$$\vec{E}^{n+1}-\vec{E}^{n}=dt\left[c\nabla\times\vec{B}^{n+1/2}\right]-dt\vec{J}^{n+1/2} \qquad (6)$$

$$\vec{B}^{n+1/2}-\vec{B}^{n-1/2}=-dt\left[c\nabla\times\vec{E}^{n}\right] \qquad (7)$$

where the superscripts $n+1/2$ and $n+1$ represent the time level. This scheme requires the use of a fully staggered grid mesh system (the Yee lattice [9]) in which $\vec{E}$ and $\vec{J}dt$ are defined at midpoints of cell-edges while the $\vec{B}$ components are defined at the midpoints of the cell-surfaces.

The trajectory of each particle, if no collision has occured, is integrated using a standard time-centering leapfrog scheme discussed in [1]. Let ii $=\gamma\vec{V}$, and the leapfrog scheme for eq(5) is written as

$$\vec{u}^{n+1/2}-\vec{u}^{n-1/2}=dt\left[\frac{q}{m}\vec{E}^{n}+\frac{\vec{u}^{n+1/2}+\vec{u}^{n-1/2}}{2\text{-f''}}\times\frac{q}{m}\frac{\vec{B}^{n}}{c}\right] \qquad (8)$$

$$\vec{z}^{n+1}-\vec{x}^{n}=\frac{\vec{u}^{n+1/2}}{\gamma^{n+1/2}}dt \qquad (9)$$

In eq(8), $\vec{E}$ and $\vec{B}$ are interpolated from the grids to the particle positions.

The code keeps track of the collision probability of each charged particle. The probability that a charged particle suffers a collision within time t is given by

$$P(t) = 1 - \exp(-\int_0^t \nu_j(t)dt) \qquad (10)$$

where ν_j is the collision frequency for the jth collisional processes. Hence, at each time step, for each particle, the following collision calculation is performed:
a) Calculate the accumulated probability: $P_{tot} = 1 - exp(-\Sigma\nu_j dt)$.
b) Chose a random number P_{ran} to decide whether a collision has occured, and the collision type.
c) If a collision has occured, calculate the new velocity of the particle after the collision from the equations for conservation of maw, momentum, and energy[3].

Implementation on MIMD Parallel Computers

The code is implemented using the General Concurrent PIC (GCPIC) algorithm developed by Liewer and Decyk[5]. The GCPIC algorithm is designed to make the most computationally intensive portion of a particle code, the particle computation, run efficiently on MIMD parallel computers. In general, the GCPIC algorithm uses two spatial decompositions of the physical domain to divide the computation efficiently among parallel processors: a primary decomposition to optimize the parallel particle push computations (i.e., particle move and current deposit) and a *secondary decomposition* to optimize the parallel field computations (i.e., field update). In the primary decomposition, each processor is assigned a subdomain and all the particles and grid points in it. Whcu a particle moves from one subdomain to another, it must be passed to the appropriate processors, which requires interprocessor communication. The primary decomposition is chosen so that both interpolations between the particles and the grids (gather/scatter) can be done *locally*, e.g., with no interprocessor communication. To ensure that the gather/scatter can be performed locally, each processor stores *guard cells, e.g.,* neighboring grid points surrounding a processor's subdomain which belong to another processor's subdomain. Fig. 1 illustrates a 2-dimensional subdomain. Interprocessor communication is necessary to exchange guard cell information.

For load balance, the primary decomposition subdomains should have roughly the same number of particles and the secondary decomposition subdomains should have the same number of grid points. When the grid is regular and the particle distribution is uniform, equal volume subdomains are optimum for both the push and field stages. Thus, for this case, the primary and secondary decompositions are identical.

In our code, the Monte Carlo collision portion of the code is completely local. Hence, the parallization of our PIC-MCC code is identical to that of our PIC code discussed in Ref[4]. In the code, the computation domain can be partitioned iuto 1-, 2-, or 3-dimensional subdomains ("slabs", "rods", or "cubes") [6]. Fig. 2 shows the flow chart of our parallel 3D electromagnetic PIC-MCC code. The main loop uses seven major subroutines. Particle Move, *Monte Carlo Collision*, Current *Deposit*, and *Field Update* for $\vec{E}$ and $\vec{B}$ (represented by the rounded blocks in Fig. 2) are the essential computation blocks in a sequential EM PIC-MCC code. On a parallel computer, each processor executes these operations independently using its own data arrays. The computations are linked together through message-passing and global communications. The code has three major message-passing subroutines: *Particle Trade, Guard Cell Summation*, and Guard Cell *Exchange* (represented by the five rectangular blocks). A detailed discussion on these message-passing subroutines can be found in Ref[4]. The global boundary conditions are also imposed in these three subroutines to avoid additional loops over grid points and particles. Currently our code uses periodic boundary conditions.

On the Delta and Paragon, we implement the message-passing based on the Express Cubix library. On the Cray T3D, we implement the message-passing using PVM.

3. Performance Analysis

To evaluate the performance of our parallel 3D EM PIC-MCC code, we use the following simple test case: two electron beams are set to counter stream in a background of fixed ions and neutrals. The electrons within each beam follow a Maxwellian distribution with thermal velocity $v_t = 0.05c$, and the drifting velocities are taken to be $v_d = \pm 0.4c$. The electrons beams interact with the neutral background through collisions. For the purpose of performance analysis, we want to keep the total number of plasma particles in the system constant. Hence, in this test case we only include elastic scattering in the MCC calculation, while other collisions are turned off.

To analyze the code performance, we measure t be total code time per time step loop T_{tot} as well as the times spent by each of the seven major subroutines for a series of runs. Let us denote T_{move}, $T_{current}$, $T_{fldupdate}$, T_{trade}, T_{gcsm}, T_{gcfl}, and T_{coll} as the time spent by *particle move, current deposit, field update,* particle *trade, current guard cell summation, field guard cell exchange,*

" . . . .

. and Monte *Carlo collision* respectively. From the measured *subroutine times, we* define the particle push time (which includes the times on moving particles, depositing currents, applying boundary conditions, and related interprocessor communications as shown in Fig. 4):

$$T^{push} = T_{move} + T_{trade} + T_{current} + T_{gcsm} \quad (11)$$

and the field solve time (which includes the times on updating the E and B fields, applying **boundary** conditions, and related interprocessor communications as shown in Fig. 4):

$$T^{field} = T_{fldupdate} + T_{gcfl} \quad (12)$$

We afso define the communication/boundary condition time for particle push as

$$T^{pushcomu} = T_{gcsm} + T_{trade} \quad (13)$$

and the total communication/boundary condition time as

$$T^{cbc} = T_{trade} + T_{gcsm} + T_{gcfl} \quad (14)$$

If only one processor is used, T'*'(1) is simply the time spent on the global boundary conditions. When multiple processors are used, $T^{cbc}(N_p > 1)$ is the sum of the time spent by the code on communications and global boundary conditions. When the number of processors used is much larger then one, T^{cb} *is* dominated by the "pure" interprocessor communication and hence, it is a good measure of the parallel communication cost [4].

As Table 1 shows, the Delta, Paragon, and Cray T3D have different memory sizes. 10 benchmark Delta, Paragon, and T3D, we compare the run times of the 3D EMPIC-MCC code on these computers in three ways: 1) same problem size per node runs, 2) same total size runs, and 3) maximum problem size runs. In *same problem size per node runs, we* load the same problem on each individual processor. In same *total* size runs, the total problem size is the same on all three computers. In *maximum problem* runs, we run the maximum problem allowed by the memory size of each computer. In all cases, we also compare the performance as a function of the processors used. Since all calculations are carried out using the default word length (i.e. 8 bytes word on T3D; 4 bytes word on Delta and Paragon), the T3D calculation doubles the precision of that on Delta and Paragon.

Same Problem Size Per Node Runs

? We first compare the performances of Delta, have Paragon, and T3D when each computer $\bigcirc$is loaded the same problem. In each run series, we keep the problem size on each individual processor fixed while increasing the total number of processors. Hence, the total problem size is proportional to the number of processors used.

An important mea-sure of the performance on a concurrent computer is the parallel efficiency ϵ which measures the effects of communication overhead and load imbalance [1O]. If there were no communications involved and the processor loads were perfectly balanced, the parallel efficiency would be $\epsilon = 100\%$. In this paper we shall focus only on the effect due to communication overhead. The simulation runs used in this paper all have near-perfect load balance because the particle distributions in these runs are nearly uniform. The parallel efficiency here is defined as

$$\epsilon(N) = \frac{T_{tot}(1)}{T_{tot}(N)N} \quad (15)$$

where $T_{tot}(N)$ is the total loop time elapsed on a parallel computer using N nodes.

We load the following problem onto Delta, Paragon, and T3D: each processor has a cubic subdomain of 16 x 16x 16 cells and 3.16x 10^5 particles ($\sim$ 77 particles/cell). This problem is scaled up in three dimensions. When this problem is loaded on 256 nodes, the total problem size becomes 64 x 128x 128 (1 .05 million) cells and ahout 80.8 million particles. When the problem is loaded on all 512 nodes of Delta and Paragon, The total problem is then 128 x 128 x 128 (2.1 million) cells and 162 million particles.

The parallel efficiencies of our code on T3D, Paragon, and Delta are shown in Fig. 3. as a function of the number of processors N_p. The results show that a high parallel efficiency of $\epsilon > 95\%$ has been achieved on all three parallel computers. Since we have perfect load balance for the test runs, the efficiency is degraded only by interprocessor communications.

Fig. 4 shows in detail the times spent by different portions of the code. In Fig. 4a, we compare the total time and the total communication time. When only 1 node is used, $T^{cbc}(1)$ consists of only the global boundary condition time. When the number of processors is $N_p > 1$, the major part of T^{cbc} is for "pure" parallel communications. We find that T^{cbc} takes less then 4% of T_{tot}. Not surprisingly, when the number of processors used is much larger than 1, $\epsilon(N_p >> 1) \simeq 1 - T^{cbc}/T_{tot}$.

In Fig. 4b, we plot the particle push time T^{push}, field solve time T^{field}, and the Monte Carlo collision time T_{coll}. In Figs. 4c and 4d, we compare the computation time and communication/boundary condition time for the particle push and the field solve respectively. Since the Monte Carlo collision part of the code is completely local, there is no communication/boundary condition time involved.

As Fig 4b shows, among the three portions of the code, the particle push is the most expansive part as expected. The field solve represents a negligible small fraction of the total time: $T^{field}/T_{tot} < 1\%$. The time spent by the Monte Carlo collision is between T^{push} and T^{field}

The particle push and field solve stage have been analysed in detail in Wang et al.[4]. Since each processor has equal number of grid cells and approximately equal number of particles, the times spent by "productive" computations, *particle* moue, current deposit, and *field update are* independent of the number of processors used. The communication/boundary condition times T_{trade}, T_{gcsm}, and T_{gcfl} increases somewhat as the node number increases. (For simulations performed here, the increase is due to communication network contention[4].) However, since they are much less than the time spent by particle moue and *current deposit*, T^{push} stays a *constant* as the number of processors is increased. Hence, the communication overbead only has a very small effect on the overall performance, In order to achieve a high efficiency on parallel computers, the problem size on each processor needs to be large enough. Wang et al. has shown that, when using 256 to 512 processors, the PIC part of the code can achieve a parallel efficiency $\epsilon \geq 95\%$ when the number of particles on each processor is $\geq 2 \times 10^5$. Even for small problems with only $\sim 5 \times 10^3$ particles/processor, the PIC part still has a parallel efficiency of $\epsilon \sim 76\%$ [4].

The Monte Carlo collision part of the code is completely local. Hence, a PIC-MCC code has an *even* better parallel efficiency than a PIC code. The Monte Carlo collision time T_{coll} depends on *the* collision frequency and time step. The collision frequency and time step used for this run are of typical values used in practical applications: $\nu = 0.1$ and $dt = 0.125$. This gives a collision probability $P = 1 - exp(-\nu dt) \simeq 0.02$, i.e. about 2 % of the totat particles undergoes a collision within each time step. For $P \simeq 2\%$, we find T_{coll} is more than an order of magnitude less than T^{push}. ($T_{coll}/T^{push} \sim 7\%$ on Delta, $T_{coll}/T^{push} \sim 6\%$ on Paragon, and $T_{coll}/T^{push} \sim 9\%$ on T3D.) We have run other simulations with $P \sim 10\%$. Even for such a high collision probability, we find T_{coll} is still only a fraction of T^{push} ($T_{coll}/T^{push} \leq 20\%$).

'he timing results plotted in Fig.4 reflect the fact that the Cray T3D has a faster CPU and network communication. For computations, even though the T3D does the calculation with 8-byte words, the speed of our code on the T3D is about 2.05 times faster than that of the Paragon, and about 2.4 times faster than that of the Delta. For interprocessor communications, the T3D is also faster ($T^{cbc}(T3D)/T^{cbc}(Paragon) \sim 0.48$, $T^{cbc}(T3D)/T^{cbc}(Paragon) \sim 0.40$) From Fig. 4, we note that the T3D also has a much less communication contention than the Delta and Paragon.

Same Total Size and Maximum Size Runs

We next compare the performances of Delta, Paragon, and T3D when each computer is loaded the same totat size or the maximum problem allowed by its memory size. When all processors are used, we find the maximum memory size that a *user's* code can occupy on each processor is approximately: 12 Mbytes (3 Mwords) per processor on Delta, 23 Mbytes (5.8 Mwords) per processor on Paragon, and 54 Mbytes (6.7 Mwords) per processor on T3D (note the word size on T3D is 8 bytes while that on Delta and Paragon *is 4* bytes).

For the study of same total size runs, we consider the following problem S]. S1 has 128 x 128 x 128 (2.1 million) cells and 162 million particles. This is the same problem we have run using all 512 processors of the Delta and Paragon in the last section. We run this problem using all three parallel computers considered here. On Delta and Paragon, this problem is run using all 512 processors. The total problem is decomposed into 8 x 8 x 8 cubic subdomains. Each subdomain has 16 x 16 x 16 cells and 3.16×10^5 particles (~ 77 particles/cell). On the Delta and Paragon, the memory required to run this problem on each node is 11.6 Mbytes, The total memory size is an equivalent of 5.9 Gbytes. This is about the largest problem that one can fit onto the Delta system. Since the T3D has only 256 processors, to fit S1 on T3D we decompose this problem into 4 x 8 x 8 subdomains. Each subdomain has 32 x 16 x 16 cells and 6.32×10^5 particles.

For maximum size runs, we consider the following two problems, S2 and S3. S2 has a domain of 104 x 168 x 168 (2.9 million) cells and 225 million particles. S2 is run only using the T3D (all 256 processors). Each processor of T3D is loaded a subdomain of 26 x 21 x 21 cells and 8.8×10^5 particles (~ 76.5 particles/cell). The memory required to run this problem *on* each processor of the T3D is 54 Mbytes. The total memory size is an equivalent of 13.8 Gbytes. This is about the largest problem that one can fit onto the 256 processor T3D.

S3 has a domain of 168 x 168 x 168 (4.74 million) cells and 361.4 million particles. S3 is run only *using* the Paragon (all 512 processors). On the Paragon, each processor is loaded a subdomain of 21^3 cells and 7.06×10^5 particles (~ 76.5 particles/cell). The memory required to run this problem on each processor of

·the Paragon *is 23.6* Mbytes. The total memory size is an equivalent of 12.1 Gbytes. This is about the largest problem that one can fit onto the 512 processor Paragon.

in Table 2, we compare the times spent by different portions of the code. For these runs, we also list collsion times for calculation s using three different collision probabilities $P \simeq 0.002$, $P \simeq 0.02$, and $P \simeq 0.095$.

one of the most important measure of the speed of the PIC portion of the code is the particle push time per particle per time step t^{push}, which is listed in Table 3. For S1, t^{push} on the three parallel computers used are:
$t^{push} \simeq$ 115 nsecs/particle/step on 512-node Delta.
$t^{push} \simeq$ 101 nsecs/particle/step on 512-node Paragon.
$t^{push} \simeq$ 105 nsecs/particle/step on 256-node T3D.
For S2 we find:
$t^{push} \simeq$ 100 nsecs/particle/step on 25 6-node T3D.
And for S3, we find:
$t^{push} \simeq$ 104 nsecs/particle/step on 512-node Paragon.

To measure the speed of the MCC portion of the code we use the collision time per collision: T_{coll}/n_{coll}. Table 3 also lists T_{coll}/n_{coll} for $P \simeq$ 0.02 (an average of 2% of total particles undergo a collision during each time step). We find T_{coll}/n_{coll} is 420 nsecs for Delta, 294 nsecs for Paragon, and 330-361 nsecs for T3D.

In Fig. 5, we compare performance of maximum problem runs on Delta, Paragon, and T3D as a function of the number of processors used or the "problem size". We plot the total run time per time step T_{tot} when running S1 on Delta, S2 on T3D, and S3 on Paragon, using different number of processors. To plot these run times on the same figure, we use a unit of "problem size" which is defined as the size for S1 on 1 node of the Delta computer, i.e. 3.16 x 10^5 particles and 16^3 grid cells. For comparison, the collision time per time step T_{coll} for $P \simeq$ 0.02 is atso plotted. Since S3 on Paragon has the largest problem size per node and S1 on Delta has the smallest problem size per node, the S3 Paragon time is the highest and the S1 Delta time is the lowest. Fig.5 shows that T_{tot} stays almost constant as both the problem size and processor number are increased. This indicates a high parallel efficiency has been achieved on all three machines.

Finally, we compare the performance of the code On MIMD parallel computers with that on a single processor Cray C90. For this study we choose one of the largest Cray C90 available, the Cray C90 at NASA Ames (The Von Neuman). The memory limit on the NASA Ames Cray C90 is 128 Mwords (1.024 Gbytes). This restricts that the largest problem we can fit on the Cray C90 for $\sim$ 77 particles/cell is 72^3 grid cells and 2.9 x 10^7 particles (or size $\simeq$ 91.13). Other than the message-passing and global communications, the C90 version of the code is identical to the parallel version. The C90 *version* of the code is compiled using tile Cray Fortran compiling system's automatic vectorization and opti miz ation. However, no xc-writing was done to optimize and vectorize the gather/scatter for the Cray. Hence, the code performance on Cray may not be the best performance one can get from a single CPU C90.

To compare the performance, we define the speedup of the three MIMI) parallel computers as

$$S = \frac{\text{(mf/.@cra,}}{(T_{tot}/size)_{MIMD}} \tag{16}$$

When the problem size is small, the Cray supercomputer performs much better than the Delta and Paragon. Compared to Cray C90, the speedup factor *at size* = 1 is *Speedup* $\simeq$ 0.12 for S2. However, as the problem size increases, the time spent on the C90 increases approximately linearly on the log state. Extrapolating the run times on C90 to the maximum problems allowed by the parallel computer, if one had a Cray C90 large enough to run these problems, then the speedup of the 51 2-processor Delta, 51 2-processor Paragon, and the 256 processor T3D over the Cray C90 would be $S_{Delta} \simeq 70$, $S_{Paragon} \simeq$ 82, and $S_{T3D} \simeq$ 84 respectively.

4. Summary and Conclusions

A MIMD parallel 3D electromagnetic PIC-MCC code has been developed. The code uses a standard relativistic leapfrog scheme to push plasma particles, a Monte Carlo scheme to calculate the collisions between plasma particles and the neutral background, and a rigorous charge-conservation finite-difference method to update electromagnetic fields. The code is parallelized using the Generat Concurrent PIC (GCPIC) algorithm[5] which uses a domain decomposition to divide the computation among the processors. 'he code is implemented on the 51 2-processor Delta and Paragon using the Express Cubix and on the 256-processor Cray T3D using PVM. It is shown that our 3D EM PIC-MCC code runs with a high parallel efficiency of $\epsilon > 95\%$ for large size problems. We have also used the 3D PIC-MCC code to benchmark Delta, Paragon, and T3D. When the problem size on each processor is identical, T3D has relatively the best performance per processor. The sizes of the maximum problems we have run on 512-processor Delta, 512-processor Paragon, and

256-processor T3D are 162 million particles(2.1 million cells), 361 million particles(4.7 million cells), and 225 million particles(2.9 million cells) respectively. Each problem was run with several million collisions per time step. For these large scale simulations, the particle push time achieved is in the range of 100-115 nsecs/particle/time step, and the collision calculation time is in the range of a few hundreds nsecs per collision. Compared with the performance on a single CPU Cray C90, this represents a factor of 70-80 speedup. We find T3D has the best performance per processor while Paragon has the best overall performance. The timing results we obtained indicate that these MIMD parallel supercomputers now allow certain large scale 3-D EM PIC-MCC simulation studies be conducted, such as simulations of plasma thruster plumes and Critical Ionization Velocity Experiments in Space.

Acknowledgments

We would like to thank V. Decyk (UCLA), S. R. Karmesin (Caltech), and R. Biasca (Northeastern) for many useful discussions. This work was carried out by the Jet Propulsion Laboratory under contracts from US Department of Energy through agreements with NASA. Access to the Delta and Paragon, which is operated by Caltech on behalf of the Concurrent Supercomputing Consortium, was provided by NASA. Access to the Cray T3D and Cray C90 was provided by funding from NASA Offices of Mission to Planet Earth, Aeronautics, and Space Science.

References

[1] C.K. Birdsall and A. B. Langdon, Plasma Physics via Computer Simulation (McGraw-Hill, New York, 1985).

[2] R.W. Hockney and J.W. Eastwood, Computer Simulation Using Particles (McGraw-Hill, New York, 1981).

[3] C.K. Birdsall, Particle-in-Cell Charged-Particle Simulations, Plus Monte Carlo Collisions With Neutral Atoms, PIC-MCC *IEEE Trans. Plasma* Science, 19 (1991), 65-85.

[4 J. Wang, P.C. Liewer, V.K. Decyk, 3D Electromagnetic Plasma Particle Simulations on a MIMD Parallel Computer to be published in *Computer Physics* Communications, 1994.

[5] P. C. Liewer and V. K. Decyk, A General Concurrent Algorithm for Plasma Particle-in-Cell Simulation Codes, *J. Computational Physics*, 85 (1989), 302-322.

[6] P.M. Lyster, P. C. Liewer, V. K. Decyk, and R. D. Ferraro, Implementation and Characterization of a 3-D Particle-in-Cell Code on MIMD Massively Parallel Supercomputers, (to be published, 1994).

[7] QUICKSILVER Programmer's Guide, Sandia National Laboratories.

[8] J. Villasenor and O. Buneman, Rigorous Charge Conservation for Local Electromagnetic Field Solvers, *Computer* Physics Communications, 69 (1992), 306-316.

[9] K. S. Yee, Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in isotropic Media, *IEEE* Trans. *Antennas Propagat., 14 (1966), 302-307.*

[10] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, Solving Problems on Concurrent Processors, Vol.I (Prentice-Hall, New Jersey, 1988)

Figure captious

Figure 1: Domain decomposition of a 2-I) simulation domain for 4 processors shosing processor subdomains and guard ceils (shaded).

Figure 2: Flow chart for the parallel 3D electromagnetic PIC-MCC code.

Figure 3: Parallel efficiency vs. number of processors used. (Solid line with square: T3D. Dashed line with circle: Paragon. Dotted line with triangle: Delta.)

Figure 4: Timing results for same problem size runs on Delta, Paragon, and T3D. (Solid line with square: T3D. Dashed line with circle: Paragon. Dotted line with triangle: Delta.)

Figure 5: Timing results for maximum problem size runs. (Solid line with square: T3D. Dashed line with circle: Paragon. Dotted line with triangle: Delta. Dash dotted line with cross: l-CPU Cray C90.)

Machine	Delta	Paragon	Cray
Model	Prototype	XPS L38	T3D
Nodes	513	512	256
Gflops	30.7	38.4	38.4
CPU	i860 XR	i860 XP(?)	DEC 21064
Speed	40 MHZ	50 MHZ	150 MHZ
Mflops/CPU			
double precision	60	75	150
single precision	80	100	
Mbytes/node	16	32	64
Total Gbytes	8.2	16.4	16.4
Topology	2D(16X36)	2D(16X36)	3D Torus

Table 1: Comparison of the hardwares of parallel supercomputers at Caltech and JPL

	particle number	grid cell number	machine	N_p	T^{push} (sec)	$T^{pushcomu}$ (sec)	T^{field} (sec)	T_{gcfl} (sec)	T_{coll} (sec) $\nu = 0.01$	T_{coll} (sec) $\nu = 0.1$	T_{coll} (sec) $\nu = 0.5$
S1	162 X 10^6	128^3	Delta	512	18.62	0.60	0.13	0.108	0.73	1.34	3.92
S1	162 x 10^6	128^3	Paragon	512	16.48	0.52	0.085	0.065	0.58	0.94	2.46
S1	162 x10^6	1283	T3D	256	17.01	0.48	0.073	0.065	0.63	1.16	3.41
S2	225 x10^6	104 x 168^2	T3D	256	22.52	0.67	0.083	0.039	0.88	1.44	3.75
S3	361.4 x 10^6	168^3	Paragon	512	37.15	1.17	0.15	0.098	1.29	2.11	5.50

Table 2: Time spent by different portions of the code for maximum problem size runs

	particle number	grid cell number	T^{push}/particle (nsecs) Delta	T^{push}/particle (nsecs) Paragon	T^{push}/particle (nsecs) T3D	n_{coll} ($\nu \simeq 0.02$)	T_{coll}/n_{coll} (nsecs) Delta	T_{coll}/n_{coll} (nsecs) Paragon	T_{coll}/n_{coll} (nsecs) T3D
S1	162 x 10^6	128^3	115	101	105	3.21 x10^6	420	294	361
S2	225 x10^6	104 x 168^2			100	4.49 x 10^6			330
S3	361.4 x 10^6	168^3		104		7.16 x 10^6		294	

Table 3: Particle push time/particle/step and Monte Carlo collision time/collision for maximum problem size runs

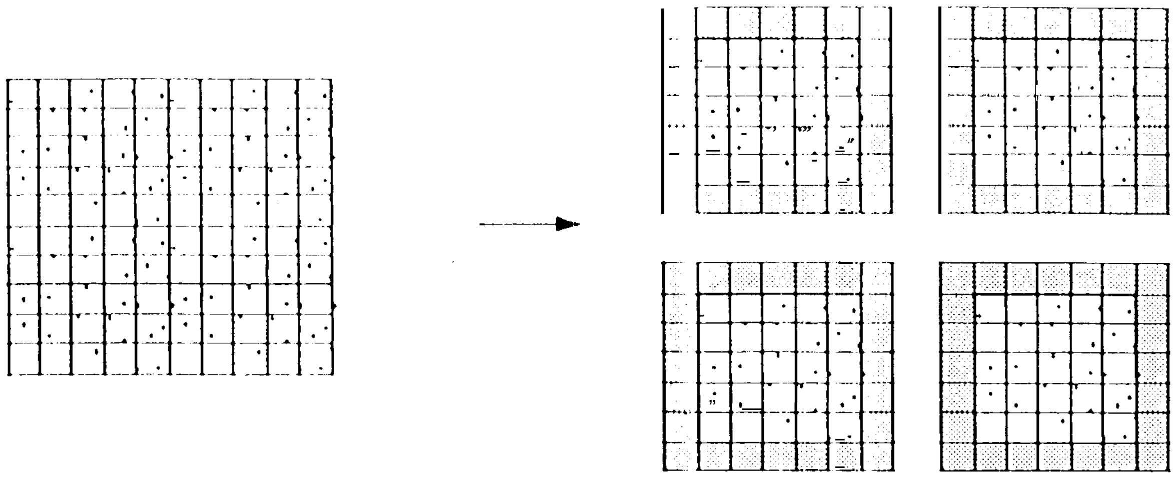


Figure 1

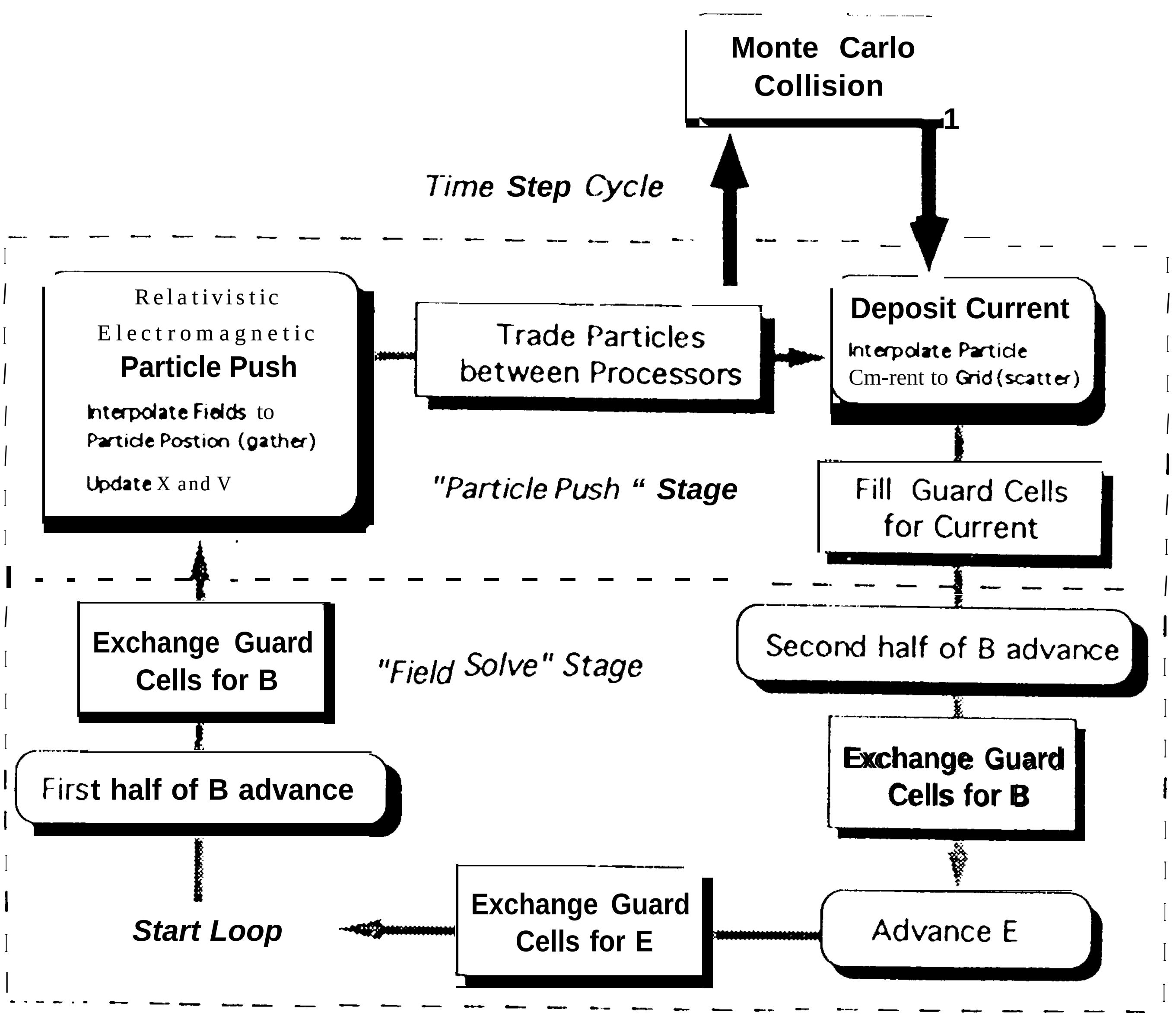

Monte Carlo Collision
Time Step Cycle
Relativistic Electromagnetic Particle Push
Interpolate Fields to Particle Postion (gather)
Update X and V
Trade Particles between Processors
Deposit Current
Interpolate Particle Cm-rent to Grid (scatter)
"Particle Push " Stage
Fill Guard Cells for Current
Exchange Guard Cells for B
"Field Solve" Stage
Second half of B advance
First half of B advance
Exchange Guard Cells for B
Start Loop
Exchange Guard Cells for E
Advance E

Figure 2

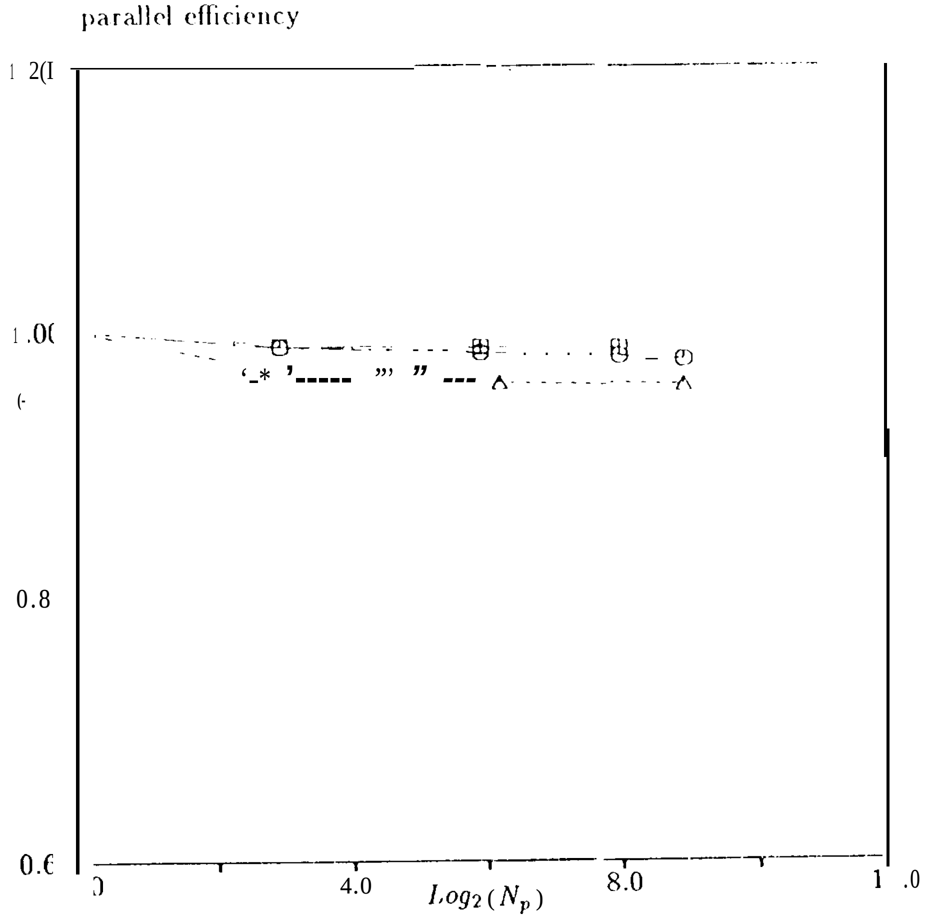
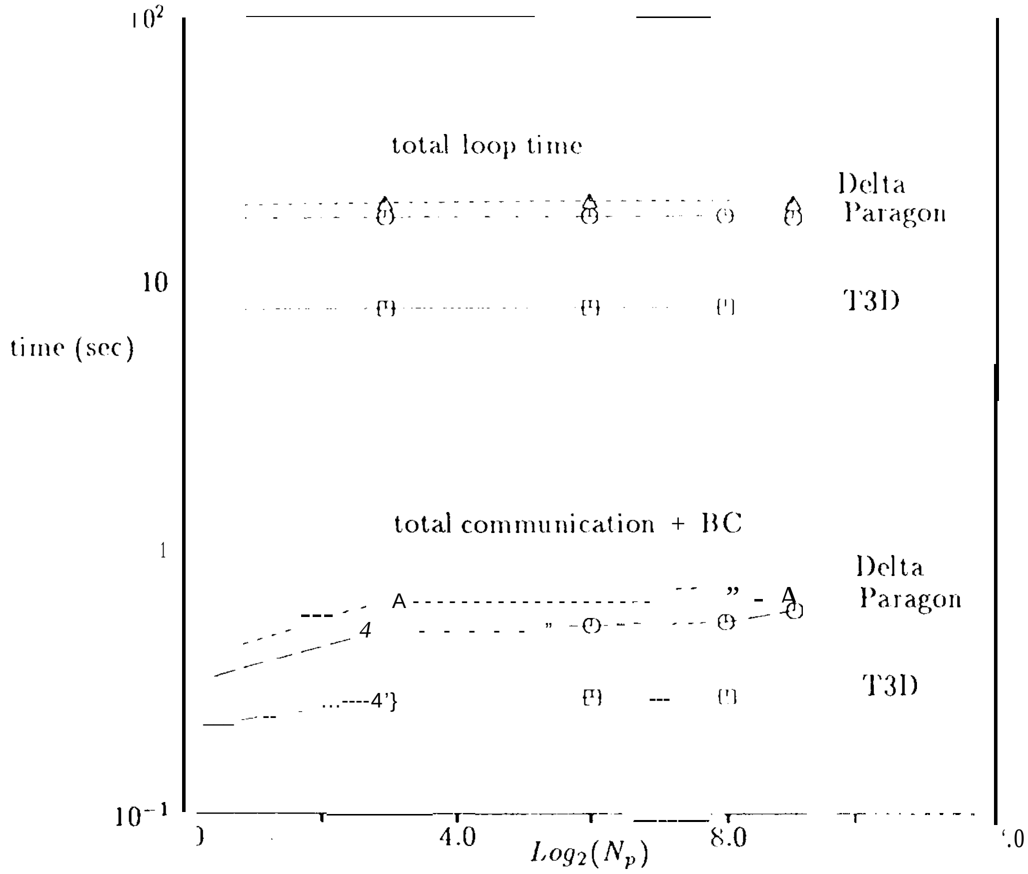


Figure 3

a) total loop and communication time



b) T^{push}, T_{mcc}, and T^{field}

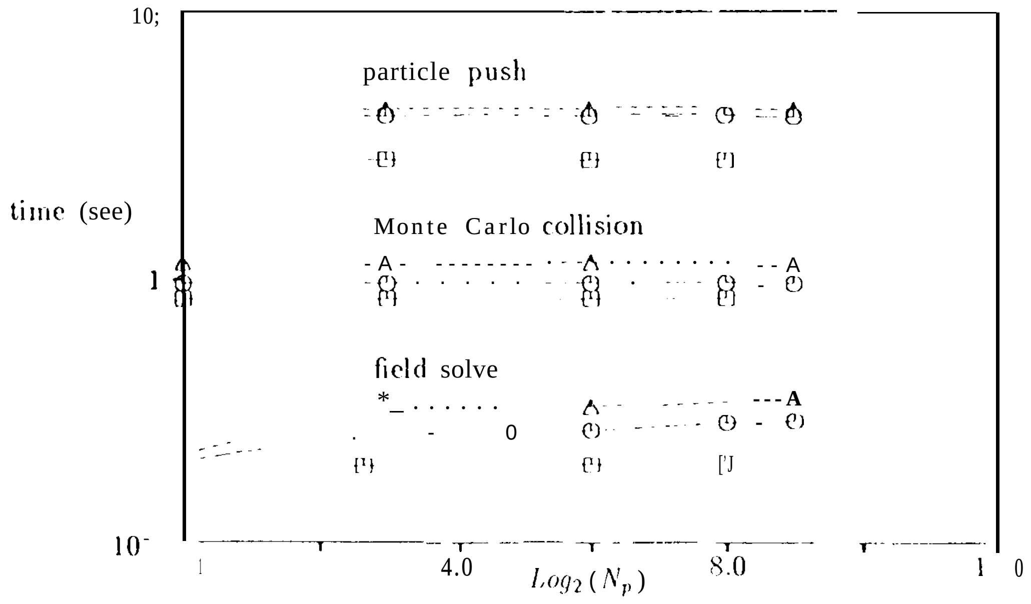
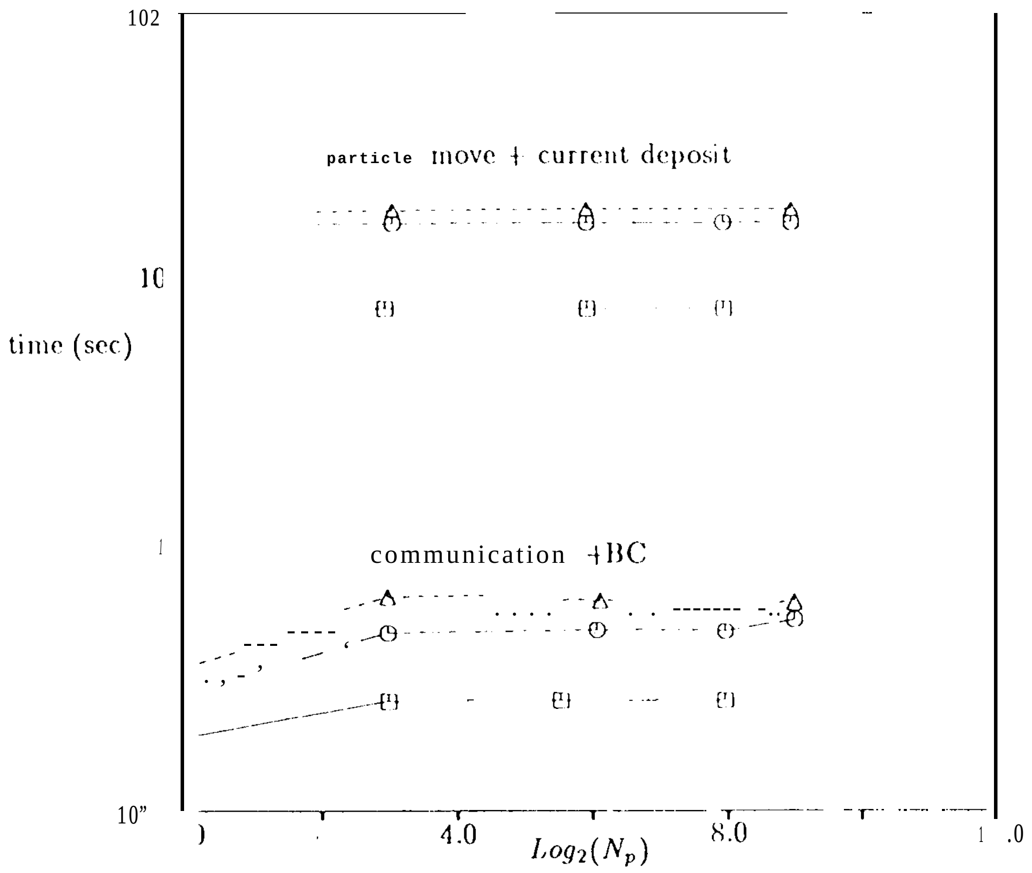


Figure 4

c) particle push



d) field solve

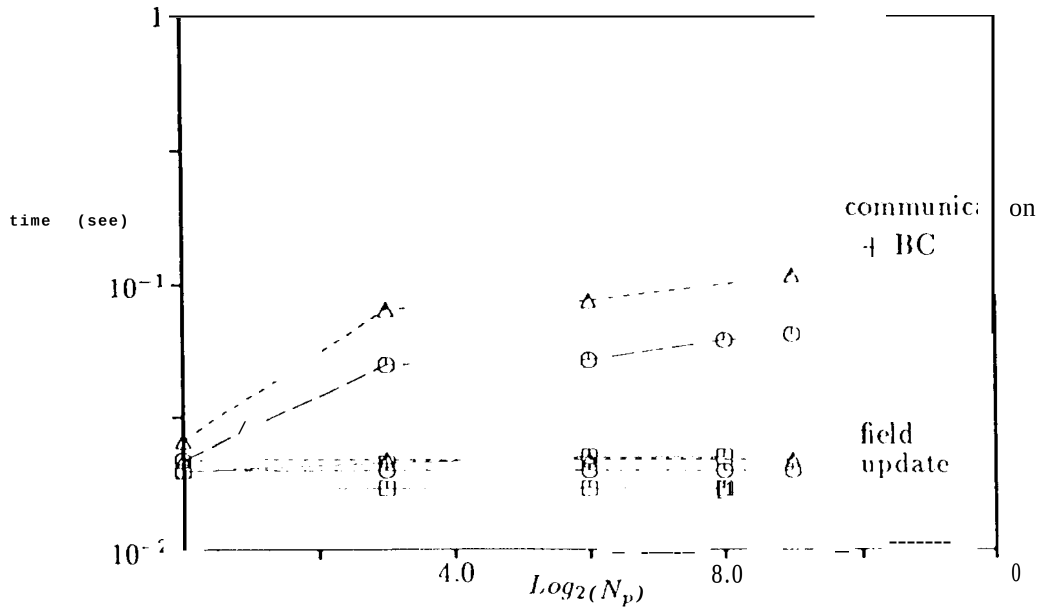
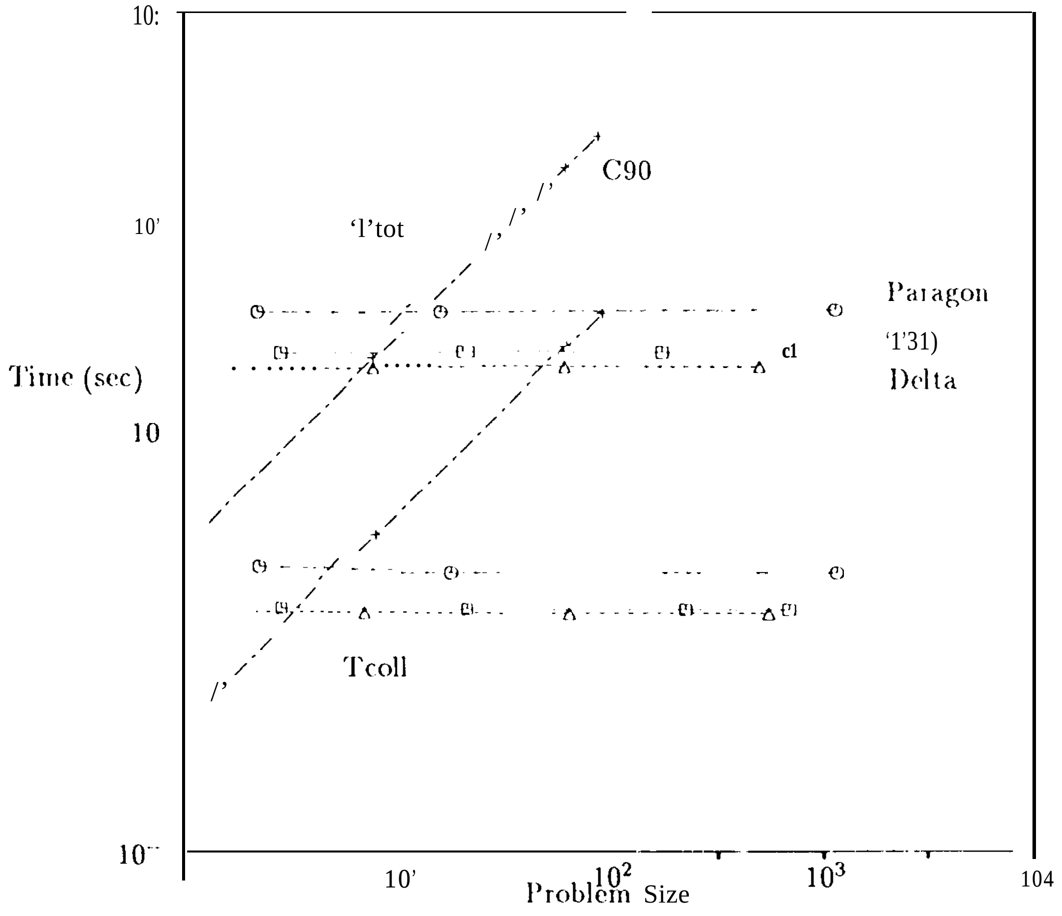


Figure 4 continue



10:
10'
10
10⁻⁻
Time (sec)
C90
'l'tot
Paragon
'1'31)
Delta
c1
Tcoll
10'
10²
10³
104
Problem Size

Figure 5